Word Embeddings, Sense Embeddings and their Application to Word Sense Induction

Linfeng Song

The University of Rochester
Computer Science Department
Rochester, NY 14627

Area Paper

April 2016

Abstract

This paper investigates the cutting-edge techniques for word embedding, sense embedding, and our evaluation results on large-scale datasets. Word embedding refers to a kind of methods that learn a distributed dense vector for each word in a vocabulary. Traditional word embedding methods first obtain the co-occurrence matrix then perform dimension reduction with PCA. Recent methods use neural language models that directly learn word vectors by predicting the context words of the target word. Moving one step forward, sense embedding learns a distributed vector for each sense of a word. They either define a sense as a cluster of contexts where the target word appears or define a sense based on a sense inventory. To evaluate the performance of the state-of-the-art sense embedding methods, I first compare them on the dominant word similarity datasets, then compare them on my experimental settings. In addition, I show that sense embedding is applicable to the task of word sense induction (WSI). Actually we are the first to show that sense embedding methods are competitive on WSI by building sense-embedding-based systems that demonstrate highly competitive performances on the SemEval 2010 WSI shared task. Finally, I propose several possible future research directions on word embedding and sense embedding.

The University of Rochester Computer Science Department supported this work.

Contents

1 Introduction

Word Embedding are a kind of methods to learn low-dimensional real-valued vectors (also called "distributed representation") for words. Previous methods learn each vector v (also called "distributional representation") by counting co-occurrence values with other words. Recent methods learns each vector v by discriminative learning that predicts the context words of the target word.

Word Embedding has been demonstrated to preserve both syntactic and semantic properties by showing that related word pairs (such as ("king","queen") and ("man","woman")) have similar vector offsets. Shown in table 1, the first example shows that the vector offset from vec("king") to vec("queen") is similar with the vector offset from vec("man") to vec("woman"). This shows the semantic regularities. By substaction, the royal properties of "king" and "queen" are canceled, and the resulting vector captures only the difference between "man" and "woman" which is the same as the substraction between "man" and "woman". Similarly, the second example demonstrates that word embedding captures the syntactic differences (from the present tense to the past tense).

vec("king") - vec("queen") = vec("man") - vec("woman")
vec("come") - vec("came") = vec("go") - vec("went")

Table 1: Syntactic and Semantic properties of word embeddings

Word embedding methods learn a single vector for each word which is problematic for polysemous words. To remedy this issue, sense embedding methods were proposed to learn a distributed representation for each sense of a word. According to the ways for defining sense, existing sense embedding methods can be roughly divided into 3 categories: clustering-based, non-parametric and ontology-based. Ontology-based methods (Chen et al., 2014; Jauhar et al., 2015; Rothe and Schütze, 2015) ground senses into a existing sense inventory such as WordNet. Clustering-based methods (Reisinger and Mooney, 2010; Huang et al., 2012a) and non-parametric methods (Neelakantan et al., 2014; Li and Jurafsky, 2015), on the other hand, treat a group of similar contexts of a word as a sense. This follows the distributional hypothesis (Harris, 1954) that the word meaning is reflected by a set of contexts where it appears. Comparatively, clustering-based methods assign fixed number of senses for each word, non-parametric methods dynamically decide the number of senses for each word by non-parametric processes, such as the Chinese Restaurant Process (CRP) (Blei et al., 2003a).

Previous methods on sense embedding primarily evaluate their performance on some context-dependent word similarity sets, such as SCWS (Huang et al., 2012a). Each instance in the testset contains two target words, two sentences in which each target word appears and 10 similarity scores assigned by human juedges. This is problematic since the testset is small and the disagreement between human graders is large (variance among the human grades is large). Table 2 shows the variances among 10 human grades on the 2003 test instances of SCWS testset. Among all the test instances, only about 4% are agreed (variance less than 1.0) and more than 37% are highly disagreed (variance greater than 10.0). The highest variance is 22.26 while the variance of integer list from 1 to 10 is only 8.25.

Variance	<1.0	1.0 to 5.0	5.0 to 10.0	<10.0
Percentage	4.15%	18.32%	40.34%	37.19%

Table 2: Percentage of variance among 10 human grades on instances of SCWS testset

We point out that the sense embedding task and the word sense induction (WSI) task are highly inter-related, and propose to evaluate sense embedding methods on WSI tasks. WSI is the task of automatically finding sense clusters for polysemous words without referring any already-known sense inventory. In detail, we evaluate the current state-of-the-art sense embedding methods on two experiments: In one experiment, we test them on SemCor corpus[1], in the other experiment we test them on some previous WSI shared tasks. Comparing to the later setting, the former setting has much larger vocabulary and more instances.

As the result, we surprisingly find that sense embedding methods generally show highly competitive performance on WSI shared tasks. We conclude that two advantages of sense embedding may result in this improvement: (1) distributed sense embeddings are taken as the knowledge representations which are trained discriminatively, and usually have better performance than traditional count-based distributional models (Baroni et al., 2014), and (2) a general model for the whole vocabulary is jointly trained to induce sense centroids under the multi-task learning framework (Caruana, 1997). We further verify the two advantages by comparing with carefully designed baselines. In addition, we find some interesting patterns by evaluating on SemCor corpus.

The remaining of this paper is organized as follows. In Chapter 2, I will introduce the general idea of word embedding, the details of two state-of-the-art word embedding methods, and my understanding of these methods. In Chapter 3, I will describe 3 directions for learning sense embeddings, then several state-of-the-art sense embedding methods under each direction, and finally their performance on SCWS testset. In Chapter 4, I will propose to evaluate sense embedding methods on WSI tasks as an alternative way. I first show that the tasks of sense embedding and WSI are highly inter-related, then analyze the results on the SemEval-2010 WSI task which recent researches on WSI commonly evaluated their works on. In Chapter 5, I will propose several future research directions on word and sense embedding. Finally, in Chapter 6, I will conclude this paper.

[1]http://moin.delph-in.net/SemCor

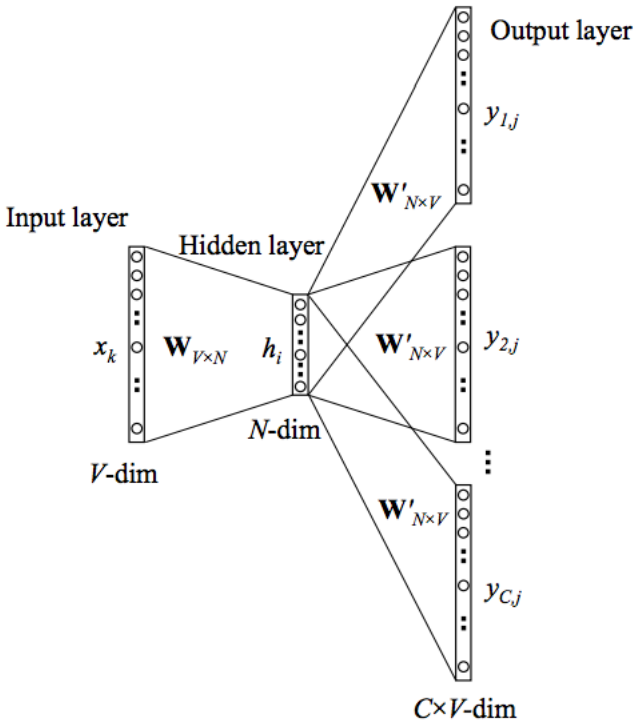

Figure 1: Neural Network structure for Skip-gram model. Figure comes from Rong (2014)

2 Word Embedding

Word embedding is a set of language modeling techniques in which words or phrases from the vocabulary are mapped to vectors of real numbers in a low-dimensional space. Traditional word embedding methods adopt dimension reduction methods (such as SVD and PCA) on the word co-occurrence matrix. Comparatively, recent methods use discriminative learning, such that the vector of the target word is updated by minimizing its distance from the vectors representing its context words and maximizing its distance from the vectors of other words. In this Chapter, I will introduce several state-of-the-art word embedding models (Skip-gram, CBOW and GloVe).

2.1 Skip-gram Model

The skip-gram model (Mikolov et al., 2013a) learns word embeddings such that they are useful for predicting the context words. As shown in Figure 1, skip-gram model is a fully connected neural network with one hidden layer. The input and output layers are vocabulary-sized sparse vectors. The input layer is one-hot representation that only the position representing the target word is 1 and the other positions are all 0s. The output layer is a probability distribution of outputing each word in the vocabulary given the input word. The reference output layer is also one-hot representation, and the errors at this layer

are calculated basing on the difference between the actual output and the reference. The hidden layer is a N-dimensional dense vector which is actually a row vector of matrix $W_{V\times N}$ (by product the input layer with the matrix). All predicting words share the same matrix $W^{'}_{N\times V}$.

Given the input word to be the ith word in the vocabulary, the probability of outputting the jth word in the context (positive case) is given by equation 1, where $vec(w_i)$ is the ith row vector of $W_{V\times N}$ and $vec(w_j)$ is the jth column vector of $W^{'}_{N\times V}$.

$$P(D=1|vec(w_i),vec(w_j)) = \frac{1}{1+e^{vec(w_i)^T vec(w_j)}} \tag{1}$$

Similarly, the probability of outputting the jth word which is not in the context (negative case) is given by equation 2.

$$P(D=0|vec(w_i),vec(w_j)) = 1 - P(D=1|vec(w_i),vec(w_j)) \tag{2}$$

Given a training set containing the sequence of word tokens $w_1, w_2, \ldots, w_T$, the word embeddings are learned by maximizing the following objective function:

$$J(W_{V\times N}, W^{'}_{N\times V}) = \sum_{i=1}^{T}\Big[\sum_{c\in C_i} P(D=1|vec(w_i),vec(c)) + \sum_{c'\in V-C_i} P(D=0|vec(w_i),vec(c'))\Big] \tag{3}$$

where $W_{V\times N}$ and $W^{'}_{N\times V}$ (mentioned before) are parameters, i is the position of the target word w_i in the sequence, C_i represents the context of word w_i, each c is a word appearing in the context of w_i and each c' is a word not appearing in the context.

To make the training faster, Stochastic Gradient Descent (SGD) is adopted on each pair of target word and context. And the error on the output layer is back-propagated through the whole network. The adapted target function is shown in equation 4, where w_i is the target word, C_i is the set of the context words and V represents the whole vocabulary.

$$J_{sgd}(W_{V\times N}, W^{'}_{N\times V}) = \sum_{c\in C_i} P(D=1|vec(w_i),vec(c)) + \sum_{c'\in V-C_i} P(D=0|vec(w_i),vec(c')) \tag{4}$$

In addition to the model, there are several techniques that have been proved helpful by later research (Levy et al., 2015). One method is called "dynamic window size". For each training word w_i, the set of context words includes N_i words to the left and right of w_i. Here N_i, the window size for w_i, is uniformly sampled from an integer list from 1 to N, where N is the manually set maximum context window size. This method highlights the influence of the surrounding words because the vector of the target word is updated more towards them comparing with the words that are relatively farther but are still within the range of N. Another method is called "sub-sampling" which is adopted to diminish the excessive influence of frequent words. Frequent words are usually function words and do not have much information for distinguishing different words. Here each word w has a

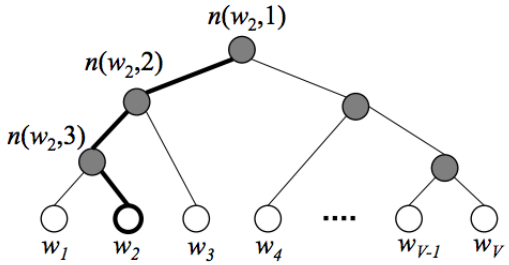

Figure 2: A binary tree for the hierarchical softmax. Figure comes from Rong (2014)

chance to be dropped out with a probability as shown in Equation 5, where $p_{unigram}(w)$ is the unigram distribution of w and Z is the normalization factor.

$$p(w) = \frac{p_{unigram}(w)^{3/4}}{Z} \tag{5}$$

Training according to the objective function in Equation 4 is impractical, since we have to enumerate all words in the vocabulary V when we sum over probabilities for the negative cases, whose size can be several millions. Here I introduce two techniques to solve this problem, which are Negative Sampling and Hierarchical Softmax. We should notice that both techniques are also applicable to the CBOW model which will be described in Section 2.2.

2.1.1 Negative Sampling

As we have shown in Equation 4, each update needs both the positive case c_j and the negative cases $V - \{c_j\}$. The idea of negative sampling is that we only need a few negative cases rather than the whole vocabulary. The negative cases are sampled without replacement from $V - \{c_j\}$ according to the probability in Equation 5. The training objective with negative sampling is shown in Equation 6 where V_{neg} is the set of negative cases sampled from $V - \{c_j\}$.

$$J_{neg} = P(D = 1|vec(w_i), vec(c_j)) + \sum_{c' \in V_{neg}} P(D = 0|vec(w_i), vec(c')) \tag{6}$$

2.1.2 Hierarchical Softmax

Hierarchical softmax, on the other hand, uses a binary tree to directly represent the probability of outputting each word in the vocabulary. Shown in Figure 2, all words in the vocabulary are the leaf nodes, and it is easy prove that there are $|V| - 1$ internal nodes. For each leaf node, there is a unique path (a sequence of steps) from the root to it. The probability of outputing that word is the product of the probability of each step.

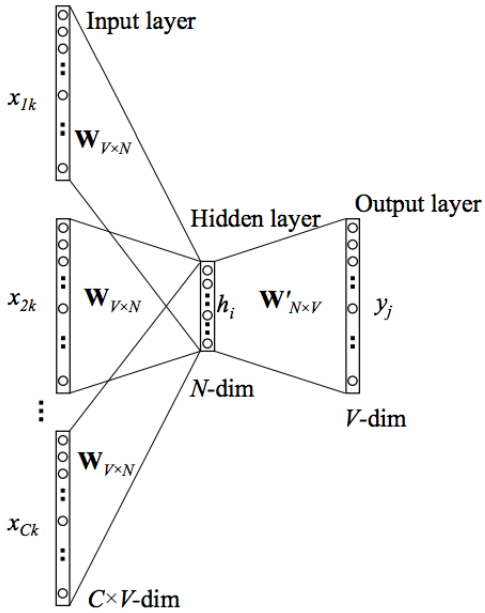

Figure 3: The Continuous Bag-of-Word (CBOW) Model. Figure comes from Rong (2014)

$$p(c_j|w_i) = \prod_{k=1}^{L(c_j)-1} \sigma\big(\mathbf{I}(n(c_j,k+1) = \mathbf{ch}(n(c_j,k)))vec(w_i)^T v'_{n(c_j,k)}\big) \tag{7}$$

In the hierarchical softmax model, there is no vector representations for words (matrix $W'_{N\times V}$). Instead, each inner node has an vector representation $v'_{n(w,k)}$. The probability of outputting word c_j given target word w_i is defined in Equation 7, where **ch** is the left child of $n(c_j,k)$, $v'_{n(c_j,k)}$ is the vector representation of inner node $n(c_j,k)$, $vec(w_i)$ is the vector representation for target word w_i and $\mathbf{I}(x)$ is an indicator that returns 1 if x is true or returns -1 otherwise. Basically hierarchical softmax encodes the probability of outputing each word into an binary tree. Starting from the root, there is a probability for going left and right at each level. So the probability of reaching a leaf node from the root is the product of the probability of the action (going left or right) at each level.

2.2 Continuous Bag-of-Word Model

Continuous Bag-of-Word (CBOW) Model is the reverse version of Skip-gram model (described in Section 2.1) that it takes a bag-of-word context as the input and outputs the target word. Shown in Figure 3, $x_{1k}, x_{2k}, \ldots, x_{Ck}$ are the input context words that each is fully connected with the hidden layer and shares the same matrix $W_{|V|\times N}$. The hidden layer is fully connected with the output layer with parameter $W'_{N\times|V|}$. Comparing with Skip-gram model which the hidden layer is just the vector of the target word, in this model the hidden layer is the average of vectors of all input words:

$$
\begin{aligned}
h &= \frac{1}{C} \cdot W \cdot (x_1, x_2, \ldots, x_C) \\
&= \frac{1}{C} \cdot (vec(x_1), vec(x_2), \ldots, vec(x_C))
\end{aligned}
\tag{8}
$$

Similar to the way I describe Skip-gram, here I define the probability w_i is the output word as:

$$
P(D = 1|w_i, h) = \frac{1}{1 + e^{v_i^T h}} \tag{9}
$$

and the probability w_i is not the output word as:

$$
P(D = 0|w_i, h) = 1 - P(D = 1|v_i, h) \tag{10}
$$

the learning target for SGD is:

$$
J_{sgd}(W, W^{'}) = P(D = 1|vec(w_i), h) + \sum_{w^{'} \in V - w_i} P(D = 0|vec(w^{'}), h) \tag{11}
$$

2.3 GloVe: Global Vectors for Word Representation

Both Skip-gram and CBOW learn word embeddings from each local word and context pair. GloVe proposes another way to learn word embeddings directly from an aggregated global word-word co-occurrence matrix. This work is based on the intuition that similar words should have similar co-occurrence numbers with other words. In this method, they first make a co-occurrence matrix basing on Equation 12 that sums over all co-occurrence instances that w_j is within the N-word window of w_j. Here $\text{dist}(\cdot, \cdot)$ is the distance function that returns the position difference between the two arguments.

$$
X_{i,j} = \sum_{w_i, w_j} \frac{1}{\text{dist}(w_i, w_j)} \tag{12}
$$

We directly show the objective of this model which is:

$$
J = \sum_{i,j} f(X_{ij})(w_i^T w_j - \log X_{ij})^2 \tag{13}
$$

where $f(X_{ij})$ is the weight function for each pair of words, and it is defined as:

$$
f(X_{ij}) = \begin{cases} (X_{ij}/X_{max})^{\alpha} & \text{if } X_{ij} < X_{max} \\ 1 & \text{otherwise} \end{cases} \tag{14}
$$

The training algorithm iterates through each element X_{ij} of co-occurrence X and do update.

2.4 Conclusion

To conclude this Chapter, I first informally define what is word embedding, then briefly compare traditional methods which count co-occurrence and the current methods which use the neural language model. Finally I introduce several popular word embedding models which are Skip-gram, CBOW and GloVe.

Personally, I think Skip-gram, CBOW and GloVe basically all learn the word embeddings by predicting the context words. They are all based on the assumption that similar words should have similar context words. All of them weight each context word by considering the distance between the target word and the context word: GloVe explicitly define the weight as $\frac{1}{\text{dist}(w,c)}$. Both Skip-gram and CBOW implicitly capture that by "dynamic window size" that nearer context words have more chance to be in the context window while farther words have less chance. However they have significant differences. The difference between Skip-gram and CBOW is the order of sequence of training instances (w_i, c_i). For Skip-gram, each target word vector is updated with all its context words at once. For CBOW, vectors of several words are updated with one context word at once. While the difference between Skip-gram and GloVe is that Skip-gram update with each local instance while GloVe update with the global co-occurrence matrix.

3 Sense Embedding

Word Embedding is problematic because the ubiquitous polysemous words harm the performance among many NLP tasks. To remedy this issue, sense embedding methods learn embeddings for senses rather than words. According to the ways for learning senses, existing sense embedding methods can be roughly divided into clustering-based, non-parametric and ontology-based. Clustering-based methods (Reisinger and Mooney, 2010; Huang et al., 2012a) learns a fixed number of senses for each polysemous word. In addition, non-parametric methods (Neelakantan et al., 2014; Li and Jurafsky, 2015) dynamically decide the number of senses by a non-parametric process. Both methods treat a group of similar contexts of a word as a sense following the distributional hypothesis (Harris, 1954) that the word meaning is reflected by a set of contexts where it appears. Finally, ontology-based methods (Chen et al., 2014; Jauhar et al., 2015; Rothe and Schütze, 2015) learn senses according to a existing sense inventory such as WordNet. In this Chapter, I will select and introduce several state-of-the-art sense embedding methods, and compare their performance on SCWS which is a context-based word similarity testset.

Intuitively, sense embedding is a fine-grain version of word embedding by clustering different contexts for each target word. For example, "Apple" can appear either with "Mac", "Windows", "iPhone" or with "juice", "pie". And the word embeddings of "Mac", "Windows" and "iPhone" are different from the word embeddings of "juice" and "pie". Both clustering-based and non-parametric sense embedding methods try to use this knowledge to separate different tokens of each target word. Ontology-based method, on the other hand, initialize sene embeddings by linguistic definitions (definition, gloss, synonyms) from a sense inventory. For example, word "bank" generally has two meanings which are "a slope of land near the water" and "a finance institute", the sense embedding can be initialized as following:

$$\begin{aligned} v_s(\text{“}bank''\text{)}_1 &= v_g(\text{“}slope''\text{)} + v_g(\text{“}land''\text{)} + v_g(\text{“}water''\text{)} \\ v_s(\text{“}bank''\text{)}_2 &= v_g(\text{“}finance''\text{)} + v_g(\text{“}institute''\text{)} \end{aligned} \tag{15}$$

3.1 Clustering-based Methods

Reisinger and Mooney (2010) makes the first attempt to learn sense embedding. Based on the standard vector-space model of word, their approach represents each word token w_i as a co-occurrence vector $vec(w_i)$ that records all unigrams occurring within the windows around w_i. To learn sense vectors, their approach first clusters all occurrences (such as w_i) of word type w into K clusters, where K is a fixed constant. Then the similarity between two word types are calculated as a function of their cluster centroids. They introduces three functions which are shown in Equations 16, 17 and 18, where $\pi_i()$ is the vector of the ith sense and $d(\cdot,\cdot)$ is the cosine distance function.

$$\text{MaxSim}(u,v) = \max_{1\le i\le K, 1\le j\le K} d(\pi_i(u), \pi_j(v)) \tag{16}$$

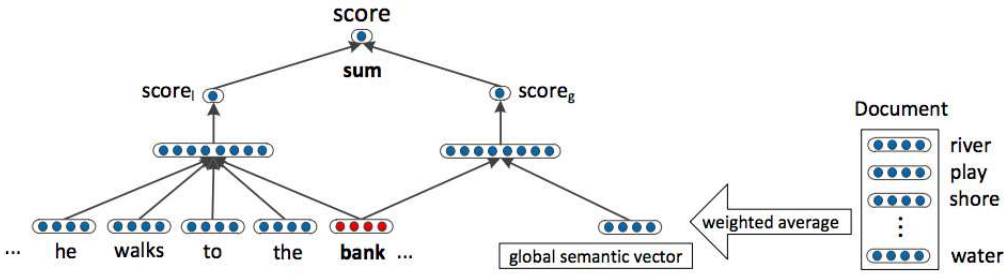

Figure 4: The recursive neural network language model. Figure comes from Huang et al. (2012b)

$$\text{AvgSim}(u, v) = \frac{1}{K^2} \sum_{i=1}^{K} \sum_{j=1}^{K} d(\pi_i(u), \pi_j(v)) \tag{17}$$

$$\text{AvgSimC}(u, v) = \frac{1}{K^2} \sum_{i=1}^{K} \sum_{j=1}^{K} d(vec(c), \pi_i(u)) \times d(vec(c^{'}), \pi_j(v)) \times d(\pi_i(u), \pi_j(v)) \tag{18}$$

Equation 16 calculates the distance between two word types by the distance of their most likely sense centroids. Comparatively, Equation 17 and 18 calculate that by averaging the distances between any pair of sense centroids. In addition, Equation 18 takes into consideration the distance between the sense and the context. In brief, Equation 18 is the weighted version of 17.

In addition, Huang et al. (2012b) proposes a neural network-based method. They first learn distributed word vectors by discriminating the next word given a word sequence with a recursive neural network. Then they cluster all occurrences of each word and re-label each occurrence according to the associated cluster. Finally, the sense embeddings are learnt by feeding the re-labeled corpus to the same neural network.

Shown in Figure 4, in their recursive neural network, the scoring components are computed by two sub neural networks, one capturing the local context and the other capturing the global context. The score of the local context uses the local word sequence which is an ordered list $s = w_1, w_2, \ldots, w_l$. And the vector $vec(s)$ equals $vec(w_1)vec(w_2) \cdots vec(w_l)$. Here $vec(w_i)$ represents the embedding of word w_i in the word sequence, and it is a column vector in the embedding matrix $W \in \mathbb{R}^{n \times |V|}$ where $|V|$ is vocabulary size and n is the vector size for each word embedding. The embedding matrix is updated during the training through standard error back-propagation process. So $score_l$ in Figure 4 is calculated via Equation 19, where $[vec(w_1)vec(w_2) \cdots vec(w_l)]$ is the concatenation of the l word embeddings, $a_1 \in \mathbb{R}^{h \times 1}$ is the content of the hidden layer after the first calculation, $W_1 \in \mathbb{R}^{h \times (ln)}$ and $W_2 \in \mathbb{R}^{1 \times h}$ are the weights of the first and second layer respectively, b_1 and b_2 are the biases, f is the element-wise activation function.

$$\begin{aligned} a_1 &= f(W_1[vec(w_1)vec(w_2)\cdots vec(w_l)] + b_1) \\ score_l &= W_2 a_1 + b_2 \end{aligned} \tag{19}$$

The score of the global context, on the other hand, uses the entire word sequence of the document, and the embedding for the document is the weighted average of word embeddings within the document as shown in Equation 20. Here, $\varphi()$ is the idf-weighting function for capturing the importance of each word in the document.

$$vec(d) = \frac{\sum_{i=1}^{|d|} \varphi(w_i) vec(w_i)}{\sum_{i=1}^{|d|} \varphi(w_i)} \tag{20}$$

In the sub neural network of the global context, the global score $score_g$ is calculated with Equation 21 which is similar with Equation 19. Here, it only concatenate the vector of the next word (which is $vec(w_l)$) and the vector of the document (which is $vec(d)$), other parameters represent similar meaning as Equation 19.

$$\begin{aligned} a_1^{(g)} &= f(W_1^{(g)}[vec(w_l); vec(d)] + b_1^{(g)}) \\ score_g &= W_2^{(g)} a_1^{(g)} + b_2^{(g)} \end{aligned} \tag{21}$$

The final score is the sum of the two scores:

$$score = score_l + score_g \tag{22}$$

Comparatively, Huang et al. (2012b) makes several improvement over Reisinger and Mooney (2010). First of all, Huang et al. (2012b) learns distributed word representation by discriminative training which has better performance than traditional counting-based methods, as shown in Baroni et al. (2014). In addition, Huang et al. (2012b) jointly learns the sense centroids for multiple word types while Reisinger and Mooney (2010) separately learns that for each word type. However Huang et al. (2012b) still suffers the error propagation problem by separating the process of labeling sense labels from the process of learning sense embeddings. More specifically, it is independent for assigning sense labels for different occurrences of a target word type. Next I will introduce Multi-Sense Skip-gram (MSSG) model of Neelakantan et al. (2014) that effeciently solves the two problems.

Generally, the MSSG model extends Skip-gram model that we have introduced in Chapter 2.1. For each occurrence of a word type, Skip-gram model updates the vector of that word type according to the context, while MSSG first finds the nearest sense to the context and updates the sense vector. More specifically, for each word type $w \in V$, it keeps a global vector $v_g(w)$, a fixed number of centroid vectors $\mu(w,k)k \in [1,..,K]$ and sense vectors $v_s(w,k)k \in [1,..,K]$.

As shown in Figure 5, for each word w_i which is surrounded by context words $c_i = [w_{i-N_i}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+Ni}]$, the context vector $v_{context}(c_i)$ is calculated by averaging

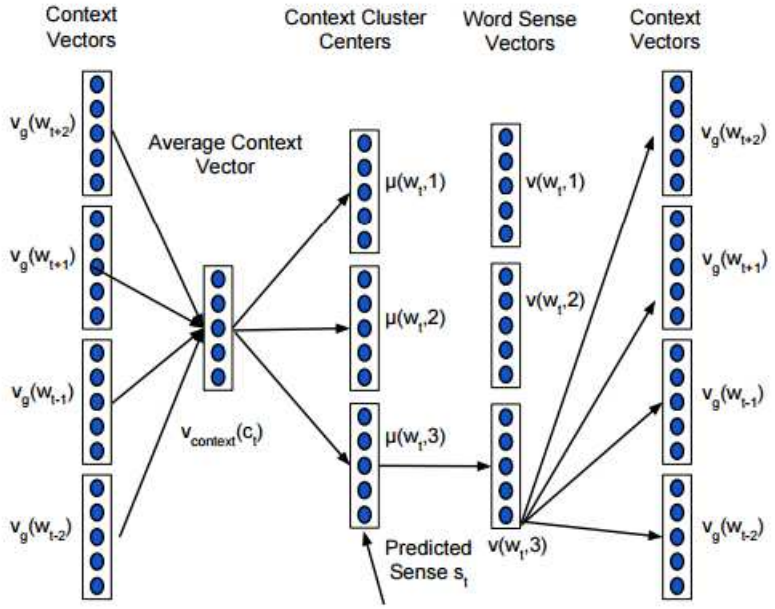

Figure 5: The framework of Multi-Sense Skip-gram (MSSG) model. Figure comes from Neelakantan et al. (2014)

the global vectors of all words within the context ($v_{context}(c_i) = \frac{1}{2N_i}\sum_{x \in c_i} v_g(x)$). Then the closest sense centroid is selected by cosine similarity as shown in Equation 23, where $\mu(w_i, k)$ is the kth centroid vector and $v_{context}(c_i)$ is the vector of context c_i. Finally both the selected centroid vector and the sense vector are updated. Here the centroid vector $\mu(w_i, k)$ is updated by adding the context vector $v_{context}(c_i)$ into it which is problematic from my point of view. Since they use k-means algorithm, but they use cosine distance rather than Euclidean distance.

$$s_i = \underset{k}{\operatorname{argmax}}\, cosine(\mu(w_i, k), v_{context}(c_i)) \qquad k \in [1, 2, \ldots, K] \tag{23}$$

For updating the sense vector, we first define the probability of observing a word c in the context in Equation 24 and the probability of not observing word $c^{'}$ in the context in Equation 25. We follow the definition style of Chapter 2.1 since MSSG is an multi-sense extension of Skip-gram model.

$$p(D = 1|v_s(w_i, s_i), v_g(c)) = \frac{1}{1 + e^{-v_s(w_i,s_i)^T v_g(c)}} \tag{24}$$

$$p(D = 0|v_s(w_i, s_i), v_g(c^{'})) = 1 - p(D = 1|v_s(w_i, s_i), v_g(c^{'})) \tag{25}$$

Given a training set containing the sequence of word tokens $w_1, w_2, \ldots, w_T$, the sense embeddings are learned by maximizing the following objective function:

$$Q = \sum_{i=1}^{T} \Big[\sum_{c \in C_i} p(D = 1 | v_s(w_i, s_i), v_g(c)) + \sum_{c' \in V - C_i} p(D = 0 | v_s(w_i, s_i), v_g(c^{'})) \Big] \tag{26}$$

where i is the position of the target word w_i in the sequence, C_i represents the context of word w_i, each c is a word appearing in the context of w_i and each c' is a word not appearing in the context.

For fast training reason, Stochastic Gradient Descent (SGD) is adopted on each pair of target word and context. And the error on the output layer is back-propagated through the whole network. The adapted target function is shown in equation 27, where w_i is the target word, C_i is the set of the context words and V represents the whole vocabulary.

$$Q_{sgd} = \sum_{c \in C_i} p(D = 1 | v_s(w_i, s_i), v_g(c)) + \sum_{c' \in V - C_i} p(D = 0 | v_s(w_i, s_i), v_g(c^{'})) \tag{27}$$

3.2 Non-parametric Sense Embedding methods

Intuitively it is problematic to assign a fixed number of senses for every word. Generally some words like "beat" have several meanings, words like "people" may have only one meaning and functional words should have only one meaning. However, clustering-based methods such as MSSG assign a equal number of senses to each word regardless of the above situations. Non-parametric processes are naturally helpful for generating a different number of senses for each word.

Probabaly the NP-MSSG model of Neelakantan et al. (2014) is the first work that addresses the task of learning a varying number of senses for different words. Generally, their method creates a new sense for a word type with the probability which is proportional to the distance from the context to the nearest sense. Initially, there is no sense for each word type, as the number of senses for each word type is unknown. Then the first sense for a word type is created from the first context in which it appears and the first context is also assigned to the first sense. After creating the first sense, the sense for each context is decided by Equation 28, where $K(w_i)$ is the current number of senses for word w_i. The context is assigned to a new sense if the distance to the nearest sense is greater than a pre-defined hyperparameter λ, or it is assigned to the nearest sense $k_{max} \in [1, 2, \ldots, K(w_i)]$ otherwise.

$$s_i = \begin{cases} K(w_i) + 1, & \text{if } \max_{k=1,2,\ldots,K(w_i)} cosine(\mu(w_i, k), v_{context}(c_i)) < \lambda \\ k_{max}, & \text{otherwise} \end{cases} \tag{28}$$

The NP-MSSG model is different from the MSSG model introduced in Chapter 3.1 only in the way of creating new senses. It also keeps a cluster vector $\mu(w_i, k)$ and a sense vector $v_s(w_i, k)$ for each sense which are updated in the same way as MSSG model. The objective function, the neural network architecture and the training algorithm are also the same that readers can refer to Chapter 3.1 for those details.

One problem NP-MSSG suffers is that it is hard to choose a proper λ which is crucial for the performance. Li and Jurafsky (2015), on the other hand, models the process of learning new senses for a word type as a Chinese Restaurant Process (CRP) (Blei et al., 2003a).

To briefly review the Chinese Restaurant Process, every data point is a customer of the restaurant in which there are several tables. Each table serves a dish, corresponding to a cluster of data points. Initially, the restaurant is empty that there is no table, and the first customer always choose to sit at a new table. The next customer either choose an existing table (share a existing cluster) or choose a new table (create a new cluster) basing on the probability distribution shown in Equation 29, where N_t is the number of customers in cluster k_t, c_i is the customer, γ is a constant hyperparameter for choosing new tables. Intuitively it is reasonable to give constant probability for opening new tables as we have no information about that table at this time.

$$p(s_i = k_t) \propto \begin{cases} N_t p(k_t|c_i), \text{if } k_t \text{ already exists} \\ \gamma, \text{if } k_t \text{ is new} \end{cases} \tag{29}$$

Back to the CRP-based sense embedding method, each CRP corresponds to a word type that whenever a customer (a token of the word type surrounded by a context) comes, the customer choose to sit either in one of the existing tables (existing senses of the word type) or in a new table (create a new sense), according to the same probability distribution defined in Equation 29, where N_t is the number of contexts assigned to sense k_t, c_i is the current context, γ is a constant hyperparameter for creating a new sense. So we can see there is a perfect match between the non-parametric sense embedding problem and CRP.

3.3 Ontology-based Methods

Despite the fact that clustering-based and non-parametric sense embedding methods have demonstrated good performance, they are not applicable to the down-streaming NLP applications that use WordNet-based senses. Another motivation for ontology-based methods is that there are well developed sense inventories (such as WordNet) containing not only the definitions and glosses but also hyponym, hypernym and synonym relations. Utilize these resources may further improve the quality of sense embeddings.

Majority ontology-based methods are roughly divided into two categories: One kind of methods use a sense inventory for initialization that they initialize sense vectors by the definition, gloss and relations between senses, then fine tune the vectors with a large plain corpus. The other kind of methods use the sense inventory as constraint that they transfer the sense embedding learning problem into a optimization problem under constraints. Here I first introduce the first kind of methods and then describe the second kind of methods.

The adapting predictive (AP) model of Jauhar et al. (2015) adapt the Skip-gram model (Mikolov et al., 2013a) by considering word senses as latent variables, and solve that by EM algorithm. Taking each word token as a observed symbol, the model first predict the sense which is a latent variable, then predict the surrounding tokens which are also observed symbols as shown in Figure 6.

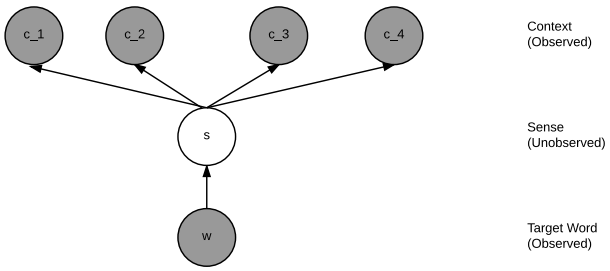

Figure 6: The generative process of adapting predictive model

To formalize the model, here I first introduce several notations which is consensus with the previous content of this survey. Let $W = \{w_1, w_2, \ldots, w_n\}$ be the whole vocabulary and $W_s = \{s_{11}, s_{12}, \ldots, s_{nm}\}$ be all the senses of the vocabulary. Let Ω be an ontology such as WordNet, then it can be formalized as a undirected graph $G_\Omega = (T_\Omega, E_\Omega)$, where each node t_{ij} corresponds to a sense s_{ij} and a edge $e_{ij-i'j'}$ connects two senses with a relation (hypernym, hyponym or synonym). $V_g = \{vec(w_i)|\forall w_i \in W\}$ is all the word embeddings and $V_s = \{vec(s_{ij})|\forall s_{ij} \in W_s\}$ is all the sense embeddings. A corpus $D = [((w_1, c_1), (w_2, c_2), \ldots, (w_n, c_n)]$ is a list of word and context pairs. As shown in Equation 30, where $p_\Omega(\theta)$ is the prior distribution of model parameter, the model maximize the joint probability of the training corpus D. At this time, the model parameter $\theta = (V_g)$ is the set of word embeddings.

$$C(\theta) = \underset{\theta}{\operatorname{argmax}}\ p_\Omega(\theta) \prod_{(w_i,c_i)\in D} p(w_i, c_i; \theta) \tag{30}$$

By considering senses, the model is reformatted as Equation 31, where the parameter θ changed to be (V_g, V_s) which is the set of both word and sense embeddings.

$$C(\theta) = \underset{\theta}{\operatorname{argmax}}\ p_\Omega(\theta) \prod_{(w_i,c_i)\in D} \sum_{s_{ij}\in s_i} p(w_i, c_i, s_{ij}; \theta) \tag{31}$$

After taking log, we get the model shown in Equation 32. Parameters can be easily updated with this equation since the two parts can be learnt independently.

$$C(\theta) = \underset{\theta}{\operatorname{argmax}}\ \log p_\Omega(\theta) + \sum_{(w_i,c_i)\in D} \log\Big(\sum_{s_{ij}\in s_i} p(w_i, c_i, s_{ij}; \theta)\Big) \tag{32}$$

The prior distribution $p_\Omega(\theta)$ is defined as the sum of Euclidean distances between each pair of senses that has a relation:

$$\log p_\Omega(\theta) \propto \gamma \sum_{ij-i'j'} ||vec(s_{ij}) - vec(s_{i'j'})||^2$$

and the joint probability $p(w_i, c_i, s_{ij}; \theta)$ is decomposed by a chain rule:

$$p(w_i, c_i, s_{ij}; \theta) = p(w_i; \theta)p(s_{ij}|w_i; \theta)p(c_i|s_{ij}; \theta)$$

Here the context c_i is independent from word w_i given the sense s_{ij} as shown in Figure 6. $p(w_i; \theta)$ is constant 1. The final form is demonstrated by Equation 33:

$$\begin{aligned} C(\theta) = \underset{\theta}{\operatorname{argmax}} \sum_{(w_i,c_i)\in D} \log\Big(\sum_{s_{ij}\in s_i} p(c_i|s_{ij};\theta)p(s_{ij}|w_i;\theta)\Big) \\ -\gamma \sum_{ij-i'j'} ||vec(s_{ij}) - vec(s_{i'j'})||^2 \end{aligned} \tag{33}$$

The model parameter θ is finally defined as (V_g, V_s, Π) that $\Pi = \{\pi_{ij}|\pi_{ij} = p(s_{ij}|w_i)\}$ is the context independent sense distribution, and V_g and V_s are the word and sense embeddings. We can see that Equation 33 can be solved by EM algorithm. However the original paper mentions that EM may lead to a poor performance, so it uses Variational Bayes to update Π as shown in Equation 34:

$$\log(\pi_{ij}^{(t+1)}) \propto \psi(\tilde{c}(w_i, s_ij) + \lambda\pi_{ij}^{(0)}) - \psi(\tilde{c}(w_i) + \lambda) \tag{34}$$

Here $\psi()$ is the digamma function which equals to $\frac{d}{dx}\ln\big(\Gamma()\big)$ and $\tilde{c}()$ is the expected count. Adopting negative sampling (Mikolov et al., 2013a), SGD on Equation 35 is used to update V_g and V_s, where c_i is the context word, $c_i^{'}$ is the negative context word, $\sigma()$ is the Sigmoid function, each $c_i^{'}$ is sampled from all negative samples. This function is similar with the objective function of Skip-gram, but with additional term to push other sense vectors $vec(s_{ij'})$ away from the current context.

$$\begin{aligned} J = \log\sigma(vec(c_i)\cdot vec(s_{ij})) + \sum_{j'\neq j} \log\sigma(-vec(c_i)\cdot vec(s_{ij'}))+ \\ \sum_m \mathbb{E}_{c_i'\sim \text{neg}}\big[\log\sigma(-vec(w_i^{'})\cdot vec(s_{ij}))\big] \end{aligned} \tag{35}$$

The overall training procedure is shown in Algorithm 1 that it iterates through the corpus D (Line 2) with online update. In the E-step, it assigns the most probable sense s_{ij} to each (w_i, c_i) (Line 3-4). In the M-step, it updates the context-independent sense distribution $\Pi^{(t+1)}$ by expected count according to Equation 34, and updates the parameters (Line 5-7).

So far I have introduced the first kind of ontology-based sense embedding methods, that utilize an sense inventory to initialize the sense vectors and then fine-tune them with a unannotated corpus. However they suffer from two drawbacks: To begin with, they do not fully utilize all the useful knowledge contained in the sense inventory. For example, Bhingardive et al. (2015) points out that information such as gloss (G), definition (D), synset members (S), gloss of the hypernymy-hyponymy synsets (HG), definition of hypernymy-hyponymy

Algorithm 1 EM training for adapting predictive model

procedure EMADAPT(D, Ω)
  for (w_i, c_i) `in` D **do**
    E-Step:
    $s_{ij} \leftarrow \text{argmax}_{s_{ij}} \pi_{ij}^{(t)} p(c_i|s_{ij}; V_g^{(t)}, V_s^{(t)})$
    M-Step:
    $\Pi^{(t+1)}$ is updated according to Equation 34
    $V_g^{(t+1)}$ and $V_s^{(t+1)}$ are updated according to Equation 35
  end for
end procedure

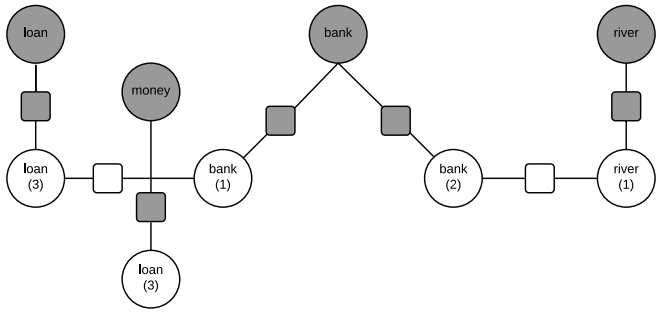

Figure 7: A example factor graph used by the Retrofitting-based method. Shaded vertices are observed. Observed variable vertices represent word types, unobserved variable vertices represent word senses. Picture comes from Jauhar et al. (2015)

synsets (HD) and synset members of hypernymy-hyponymy synsets (HS) is helpful for learning high quality sense embeddings for the WSD task. Jauhar et al. (2015) only utilize information from S and HS and Chen et al. (2014) uses knowledge from D and G. In addition, they are relatively time consuming as they iterate through a large corpus.

Comparatively, the second kind of ontology-based methods that use an sense inventory as constraint do not suffer from these drawbacks, as they use more knowledge to make constraints and directly solve the optimization problem that do not iterate through a unannotated corpus. Here let me introduce two methods of this kind: First of all, the Retrofitting-based approach of Jauhar et al. (2015) learns sense embeddings through a factor graph representing a sense inventory, such as WordNet. In the graph, every variable vertex either represents a word or represents a sense, every factor vertex connects either a sense and a word meaning the word has the sense, or two senses meaning there is a relation between them. Obviously the minimum tree width of this graph is 2 so the time complexity for optimization with it is small. Figure 7 shows an example of the factor graph. We can see that word "bank" is associated with two senses "bank(1)" which is related with sense "loan(3)" of word "loan" and sense "money(1)" of word "money", and "bank(2)" which is related with sense "river(1)" of word "river".

Following the same definition of Chapter 3.3, the potential for each clique (which is

Algorithm 2 Coordinate descent for Retrofitting-based method

procedure CoorRetro(Ω)
  while $||vec(s_{ij})^{(t)} - vec(s_{ij})^{(t-1)}|| \geq \epsilon, \ \forall s_{ij} \in T_\Omega$ **do**
    for $s_{ij} \in T_\Omega$ **do**
      $vec(s_{ij})^{(t+1)}$ is updated according to Equation 37
    end for
  end while
end procedure

	Synset_1	...	Synset_i	...	Synset_n
people	1	...		...	
...		...		...	
dog		...	1	...	
...		...		...	
hound		...	1	...	
...		...		...	

Table 3: Representing WordNet as a sparse matrix.

always pairwise) set as $e^{||u-v||^2}$. Their optimization prolbem is to find the maximum a posterior (MAP) estimation of V_s given V_g, so the optimization problem is:

$$C(V_s) = \underset{V_s}{\text{argmin}} \sum_{i-ij} \alpha ||vec(w_i) - vec(s_{ij})||^2 + \sum_{ij-i'j'} \beta_r ||vec(s_{ij}) - vec(s_{i'j'})||^2 \tag{36}$$

where both α and β_r are coefficients and β_r is relation specific. It is not hard to get the analytical solution of Equation 36 which is:

$$vec(s_{ij}) = \frac{\alpha vec(w_i) + \sum_{i'j' \in \{ij-i'j'\}} \beta_r vec(s_{i'j'})}{\alpha + \sum_{i'j' \in \{ij-i'j'\}} \beta_r} \tag{37}$$

And coordinate descent is adopted to iteratively update the parameters (sense embeddings) using Equation 37. This procedure is demonstrated in Algorithm 2.

Finally in this Chapter, let me introduce *AutoExtend* which received the best student paper in ACL-15. To better understand this work, let me first briefly introduce WordNet. WordNet can be seen as a sparse matrix that each row represents a word, each column represents a Synset and each cell is a lexeme representing a sense. A Synset is a set of synonymous senses. A word can have multiple senses, and is associated with a Synset if there is a sense that is associated with both of them. Table 3 visualize an example where both word "dog" and word "hound" are associated with Synset_i, so they have similar senses.

Generally, this work learns sense embeddings by AutoEncoders that first take a word embedding matrix as input, then compress it into a Synset embedding matrix, finally reproduce the word embedding matrix by transforming from the Synset embedding matrix back to the word embedding matrix. I demonstrate the model with Figure 8. Here the input

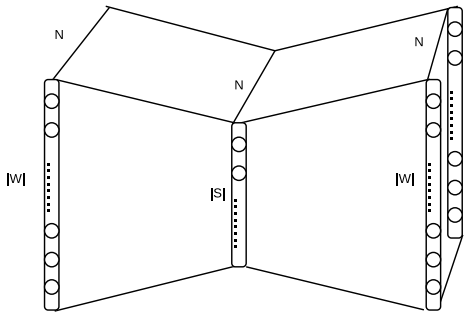

Figure 8: The neural network structure for sense embedding learning.

layer of the Autoencoder is a $|W| \times N$ sized matrix, and the hidden layer is a $|S| \times N$ sized matrix.

More formally, let $W \in \mathbb{R}^{|W| \times N}$ be the word matrix and $S \in \mathbb{R}^{|S| \times N}$ be the Synset matrix, D and $E \in \mathbb{R}^{|W| \times |S| \times N \times N}$ are two 4 dimensional tensors that E is the parameter to encode the word matrix into the synset matrix and D is the parameter to decode the Synset matrix into the word matrix. Overall the optimization function is shown in Equation 38, where $\bigotimes$ is tensor multiplication.

$$\underset{D,E}{\operatorname{argmin}} ||D \bigotimes E \bigotimes W - W|| \tag{38}$$

Given any specific pair $w^{(i)} \in W$ and $s^{(j)} \in S$, $E^{(i,j)}$ and $D^{(j,i)} \in \mathbb{R}^{N \times N}$ are the encoding and decoding parameters concerning about the pair.

$$\begin{aligned} l^{(i,j)} &= E^{(i,j)} w^{(i)} \\ l^{(i,j)} &= D^{(j,i)} s^{(j)} \end{aligned} \tag{39}$$

As shown in Equation 39, we can get the embedding of a specific lexeme $l^{(i,j)}$ by either multiplying $E^{(i,j)}$ with the related word embedding $w^{(i)}$ or multiplying $D^{(j,i)}$ with the related Synset embedding $s^{(j)}$. Here it makes two important assumptions that the sum of all lexeme embeddings belonging to the word is equal to the word embedding, and similarly the sum of all lexeme embeddings associated with the Synset is equal to the Synset embedding:

$$\begin{aligned} w^{(i)} &= \sum_j l^{(i,j)} \\ s^{(j)} &= \sum_i l^{(i,j)} \end{aligned} \tag{40}$$

By combining Equation 39 and Equation 40, we have one of the two very important constraints under which we solve the optimization:

$$\sum_j E^{(i,j)} = I_N \tag{41}$$

And similarly the other constraint is:

$$\sum_i D^{(j,i)} = I_N \tag{42}$$

Here we further assumes $E^{(i,j)}$ and $D^{(i,j)}$ are diagonal matrix, so optimizing each dimension $d \in [1, \ldots, N]$ is independent from each other. We further refine the two constraints as Equation 43 and 44, where $R(w_i, s_j) = 0$ represents there is no relationship between word w_i and Synset s_j. The second function of each Equation means that the element is 0 if the dimension indices are different or the word does not have a lexeme that is a member of the Synset.

$$\begin{aligned} &\sum_j E_{i,j,d_1,d_2} = 1, \ \forall i, d_1, d_2 \\ &E_{i,j,d_1,d_2} = 0, \text{ if } d_1 \neq d_2 \text{ or } R(w_i, s_j) = 0 \end{aligned} \tag{43}$$

$$\begin{aligned} &\sum_i D_{j,i,d_1,d_2} = 1, \ \forall j, d_1, d_2 \\ &D_{j,i,d_1,d_2} = 0, \text{ if } d_1 \neq d_2 \text{ or } R(w_i, s_j) = 0 \end{aligned} \tag{44}$$

I think there are two advantages for making this assumption which are: this makes model training faster, and this avoids over-fitting. Finally, they refine the original learning target (Equation 38) as 3 objectives,

$$\text{argmin}_{D^{(d)}, E^{(d)}} ||D^{(d)} E^{(d)} w^{(d)} - w^{(d)}|| \ \ \forall d \tag{45}$$

$$\text{argmin}_{D^{(d)}, E^{(d)}} ||E^{(d)} diag(w^{(d)}) - D^{(d)} diag(s^{(d)})|| \ \ \forall d \tag{46}$$

$$\text{argmin}_{E^{(d)}} ||R E^{(d)} w^{(d)}|| \ \ \forall d \tag{47}$$

where $R \in \mathbb{R}^{r \times |S|}$ is the WordNet relation matrix and r is the number of related Synsets. All of the 3 objectives are about only one dimension d: Equation 45 models the same target as Equation 38 that both equations minimize the information lost after encoding and decoding through out the AutoEncoder neural network in Figure 8. Equation 46 try to impose lexeme constraints that we should get the same lexeme embeddings both by going from word embeddings and by going from Synset embeddings. Equation 47 captures additional relations between Synsets such as hypernym-hyponym, antonym and verb group. Intuitively, Synsets having relations should have similar embeddings. All of the relations can be extracted from WordNet without additional refinement. To learn with all of the 3 objectives, they use linear interpolation with weights α, β and $1 - \alpha - \beta$.

Model	MaxSim	AvgSim	AvgSimC
Huang	26.1	62.8	65.7
MSSG	57.26	67.2	69.3
MSSG-NP	59.80	67.3	69.1
CRP	66.4	-	67.0
Retro	-	-	41.7
EM	-	-	61.3
Retro+EM	-	-	58.7
AutoExtend	-	68.9	**69.8**

Table 4: Representing WordNet as a sparse matrix.

3.4 Evaluating on SCWS testset

Existing works mainly use SCWS dataset (Huang et al., 2012a) for similarity evaluation. It contains 2003 instances each of which contains two target words and 10 human grades on a 10-point scale representing the similarity between them. SCWS dataset contains not only the isolated target words, the human annotated similarity scores, but also a context for each word. To evaluate on SCWS dataset, we first calculate the similarity score for each instance, then adopt Spearman correlation (ρ) between the similarity vector and the reference vector of averaged human scores. There are several ways, such as MaxSim (Equation 16), AvgSim (Equation 17) and AvgSimC (Equation 18) to calculate the similarity score.

We collect the performance scores from each paper and show them in table 4. Here *Huang* stands for the method of Huang et al. (2012a), *MSSG* and *MSSG* represents the corresponding models of Neelakantan et al. (2014), *CRP* is the work of Li and Jurafsky (2015), *EM* and *Retro* corresponds to the two methods mentioned in Jauhar et al. (2015) and *Retro+EM* represents running *Retro* to get the initial embedding of *EM*, *AutoExtend* is the Autoencoder-based method of Rothe and Schütze (2015). First of all, we can see that AvgSimC is the best among the three inference algorithms, while MaxSim is the worst. I think it is because MaxSim considers the least information which is the pair of the most probable senses. While AvgSim considers all possible pairs of senses, plus AvgSimC consider all possible pairs of senses weighted by their similarity with the current context. In addition, clustering-based method (MSSG) shows comparative result with non-parametric method (MSSG-NP). Even though intuitively different words should have different number of senses. I think we may need to refine functional words with more senses in order to company their varing context. Further more, CRP shows much higher score on MaxSim than the others. I think it is because the "rich get richer" property of CRP makes it conservative for assigning senses other than the most frequent sense, and usually simply assigning the most frequent sense can get a high performance. Finally, AutoExtend shows the best performance (in bold), proving the soundness of its method.

3.5 Evaluating on SemCor corpus

Evaluating sense embeddings is hard as there is no gold answer for each sense vector. As stated in Chapter 1, previous works mainly compare on SCWS dataset (Huang et al., 2012a)

whose size is small and the disagreement between human grades is large. For the size, the SCWS dataset only contains 2003 instances. For the agreement between human grades, we calculate the variance of the 10 grades for each instance which shows that only 4% are less than 1.0 while more than 37% are greater than 10.0. The highest variance is 22.26 while the variance of integers from 1 to 10 is only 8.25.

Here we propose to evaluate sense embedding methods on SemCor corpus[2]. SemCor is a sense annotated news domain corpus. It contains round 30k sentences, and all senses are according to WordNet. To evaluate on SemCor, one sense embedding model labels each word with a sense label, then compare the results with the gold reference. For unsupervised methods, we first use Hungarian algorithm (Kuhn, 1955) to do the matching. SemCor has two advantages from SCWS: 1) SemCor has gold reference and the evaluation is precise (either right or wrong). 2) SemCor contains much more word types and instances than SCWS that evaluating on more instances is generally more convincing.

We use 2 more indices in addition to precision: Variation of Information and Paired F-score. Variation of Information (VI) (Meilă, 2003) provides additional information besides labeling agreement (Goldwater and Griffiths, 2007), such as two results which have identical labeling precision with reference can have very different VI result, so we also investigate the VI between any two induction results. We calculate basing on Equation 48 such that r_{ij} is the probability for choosing the intersection between cluster i from one result and cluster j from the other, p_i and q_j represents the probability for choosing points in cluster i from one result and cluster j from the other result respectively.

$$VI = -\sum_{i,j} r_{ij}[\log(r_{ij}/p_i) + \log(r_{ij}/q_j)] \tag{48}$$

Overall the VI between two clustering results X and Y is the amount of information lost and the amount to be gained by changing from X to Y which is shown in Equation 49. So if two clustering results are identical, the VI between them is 0. Otherwise it is greater than 0. The more different they are, the larger VI is.

$$VI(X;Y) = H(X) + H(Y) - 2I(X,Y) \tag{49}$$

Different from other metrics, paired F-score evaluates two clustering results by transforming this into a classification problem. For each cluster c_i in either result, we generate every instance pairs from it $\binom{c_i}{2}$. Let $F(X)$ and $F(Y)$ be the set of instance pairs exist in result X and Y respectively, precision and recall are shown in Equation 50 and paired F-score is the harmonic mean $(\frac{2pr}{p+r})$ between them.

$$\begin{aligned} p &= \frac{F(X) \cap F(Y)}{F(X)} \\ r &= \frac{F(X) \cap F(Y)}{F(Y)} \end{aligned} \tag{50}$$

[2]http://moin.delph-in.net/SemCor

	CRP	MSSG	AutoExtend	OneSense
MFS(%)	79.33	68.46	85.11	100.0
Precision(%)	59.90	57.65	62.89	67.51
VI	1.22	1.31	1.11	0.88
Paired-F(%)	57.81	53.65	64.28	65.09

Table 5: Representing WordNet as a sparse matrix. *MFS* stands for most frequent sense, VI is Variation of Information.

We choose AutoExtend (Rothe and Schütze, 2015), the MSSG model of Neelakantan et al. (2014) and the CRP-based method of Li and Jurafsky (2015) for comparing. It is because they provides implementation for their work and reported the state-of-the-art performance in their papers. We also add baseline *OneSense* that always assign the same sense since previous sense disambiguation and induction tasks always include that baseline for comparison. We choose snapshot of Wikipedia 2010 as the training data which contains around 2 million documents and 990 million tokens. We select all word types with more than 100 occurrences from SemCor as the test data. As a result, our test data contains 15,469 word types with 193,118 instances. Comparing with SCWS, our test set has much more test instances.

I show the performance scores in Table 5. Here *MFS(%)* represents the percentage of the most frequent sense, *Precision(%)* is the percentage of instances that is labeled correctly comparing with the gold reference. First of all, we can see that *OneSense* shows the best performance on each index. Previous shared tasks on sense disambiguation and induction also observed this phenomenon (Agirre and Soroa, 2007; Manandhar et al., 2010). However we still observe that other systems show pretty close performance on the index of Paired F-score. The *MFS(%)* of golden is 67.51 which is the *Precision(%)* of *OneSense*, and *MSSG* shows the closest *MFS(%)* number with that. Finally, we can see that the performances on all other indices are proportional to the number on *MFS(%)*.

4 Sense Embedding for WSI

4.1 Introduction and Overview

Word sense induction (WSI) is the task of automatically finding sense clusters for polysemous words. In contrast, word sense disambiguation (WSD) assumes there exists an already-known sense inventory, and the sense of a word type is disambiguated according to the sense inventory. Therefore, clustering methods are generally applied in WSI tasks, while classification methods are utilized in WSD tasks. WSI has been successfully applied to many NLP tasks such as machine translation (Xiong and Zhang, 2014), information retrieval (Navigli and Crisafulli, 2010) and novel sense detection (Lau et al., 2012).

However, existing methods usually represent each instance with discrete hand-crafted features (Bordag, 2006; Chen et al., 2009; Van de Cruys and Apidianaki, 2011; Purandare and Pedersen, 2004), which are designed manually and require linguistic knowledge. Most previous methods require learning a specific model for each polysemous word, which limits their usability for down-stream applications and loses the chance to jointly learn senses for multiple words.

There is a great advance in recent distributed semantics, such as word embedding (Mikolov et al., 2013b; Pennington et al., 2014) and sense embedding (Reisinger and Mooney, 2010; Huang et al., 2012a; Jauhar et al., 2015; Rothe and Schütze, 2015; Chen et al., 2014; Tian et al., 2014). Comparing with word embedding, sense embedding methods learn distributed representations for senses of a polysemous word, which is similar to the sense centroid of WSI tasks.

In this work, we point out that the WSI task and the sense embedding task are highly inter-related, and propose to jointly learn sense centroids (embeddings) of all polysemous words for the WSI task. Concretely, our method induces several sense centroids (embedding) for each polysemous word in training stage. In testing stage, our method represents each instance as a contextual vector, and induces its sense by finding the nearest sense centroid in the embedding space. Comparing with existing methods, our method has two advantages: (1) distributed sense embeddings are taken as the knowledge representations which are trained discriminatively, and usually have better performance than traditional count-based distributional models (Baroni et al., 2014), and (2) a general model for the whole vocabulary is jointly trained to induce sense centroids under the multi-task learning framework (Caruana, 1997). Evaluated on SemEval-2010 WSI dataset, our method outperforms all participants and most of the recent state-of-the-art methods.

4.2 Methodology

4.2.1 Word Sense Induction

WSI is generally considered as an unsupervised clustering task under the distributional hypothesis (Harris, 1954) that the word meaning is reflected by the set of contexts in which it appears. Existing WSI methods can be roughly divided into feature-based or Bayesian. Feature-based methods first represent each instance as a context vector, then utilize a

clustering algorithm on the context vectors to induce all the senses. Bayesian methods (Brody and Lapata, 2009; Yao and Van Durme, 2011; Lau et al., 2012; Goyal and Hovy, 2014; Wang et al., 2015), on the other hand, discover senses based on topic models. They adopt either the LDA (Blei et al., 2003b) or HDP (Teh et al., 2006) model by viewing each target word as a corpus and the contexts as pseudo-documents, where a context includes all words within a window centered by the target word. For sense induction, they first extract pseudo-documents for the target word, then train topic model, finally pick the most probable topic for each test pseudo-document as the sense.

All of the existing WSI methods have two important factors: 1) how to group similar instances (clustering algorithm) and 2) how to represent context (knowledge representation). For clustering algorithms, feature-based methods use k-means or graph-based clustering algorithms to assign each instance to its nearest sense, whereas Bayesian methods sample the sense from the probability distribution among all the senses for each instance, which can be seen as soft clustering algorithms. As for knowledge representation, existing WSI methods use the vector space model (VSM) to represent each context. In feature-based models, each instance is represented as a vector of values, where a value can be the count of a feature or the co-occurrence between two words. In Bayesian methods, the vectors are represented as co-occurrences between documents and senses or between senses and words. Overall existing methods separately train a specific VSM for each word. No methods have shown distributional vectors can keep knowledge for multiple words while showing competitive performance.

4.2.2 Sense Embedding for WSI

As mentioned in Section 1, sense embedding methods learn a distributed representation for each sense of a polysemous word. There are two key factors for sense embedding learning: (1) how to decide the number of senses for each polysemous word and (2) how to learn an embedding representation for each sense. To decide the number of senses in factor (1), one group of methods (Huang et al., 2012a; Neelakantan et al., 2014) set a fixed number K of senses for each word, and each instance is assigned to the most probable sense according to Equation 51, where $\mu(w_t, k)$ is the vector for the k-th sense centroid of word w, and v_c is the representation vector of the instance.

$$s_t = \arg\max_{k=1,..,K} sim(\mu(w_t, k), v_c) \tag{51}$$

Another group of methods (Li and Jurafsky, 2015) employs non-parametric algorithms to dynamically decide the number of senses for each word, and each instance is assigned to a sense following a probability distribution in Equation 52, where S_t is the set of already generated senses for w_t, and γ is a constant probability for generating a new sense for w_t.

$$s_t \sim \begin{cases} p(k|\mu(w_t, k), v_c) \ \forall \ k \in S_t \\ \gamma \quad \text{for new sense} \end{cases} \tag{52}$$

From the above discussions, we can obviously notice that WSI task and sense embedding task are inter-related. The two factors in sense embedding learning can be aligned to the

Algorithm 3 Sense Embedding Learning for WSI

```
1: procedure TRAINING(Corpus C)
2:     for iter in [1..I] do
3:         for w_t in C do
4:             v_c ← context_vec(w_t)
5:             s_t ← sense_label(w_t, v_c)
6:             update(w_t, s_t)
7:         end for
8:     end for
9: end procedure
```

two factors of WSI task. Concretely, deciding the number of senses is the same problem as the clustering problem in WSI task, and sense embedding is a potential knowledge representation for WSI task. Therefore, sense embedding methods are naturally applicable to WSI.

In this work, we apply the sense embedding learning methods for WSI tasks. Algorithm 3 lists the flow of our method. The algorithm iterates several times over a Corpus (Line 2-3). For each token w_t, it calculates the context vector v_c (Line 4) for an instance, and then gets the most possible sense label s_t for w_t (Line 5). Finally, both the sense embeddings for s_t and global word embeddings for all context words of w_t are updated (Line 6). We introduce our strategy for *context_vec* in the next section. For *sense_label* function, a sense label is obtained by either Equation 51 or Equation 52. For the *update* function, vectors are updated by the Skip-gram method (same as Neelakantan et al. (2014)) which tries to predict context words with the current sense. In this algorithm, the senses of all polysemous words are learned jointly on the whole corpus, instead of training a single model for each individual word as in the traditional WSI methods. This is actually an instance of multi-task learning, where WSI models for each target word are trained together, and all of these models share the same global word embeddings.

Comparing to the traditional methods for WSI tasks, the advantages of our method include: 1) WSI models for all the polysemous words are trained jointly under the multi-task learning framework; 2) distributed sense embeddings are taken as the knowledge representations which are trained discriminatively, and usually have better performance than traditional count-based distributional models (Baroni et al., 2014). To verify the two statements, we carefully designed comparative experiments described in the next section.

4.3 Experiment

4.3.1 Experimental Setup and baselines

We evaluate our methods on the test set of the SemEval-2010 WSI task (Manandhar et al., 2010). It contains 8,915 instances for 100 target words (50 nouns and 50 verbs) which mostly come from news domain. We choose the April 2010 snapshot of Wikipedia (Shaoul and Westbury, 2010) as our training set, as it is freely available and domain general. It contains around 2 million documents and 990 million tokens. We train and test our models and the

System	V-Measure(%)			Paired F-score(%)			80-20 SR(%)			FS	#CI
	All	Noun	Verb	All	Noun	Verb	All	Noun	Verb	All	
UoY (2010)	15.7	20.6	8.5	49.8	38.2	66.6	62.4	59.4	66.8	-	11.54
NMF_{lib} (2011)	11.8	13.5	9.4	45.3	42.2	49.8	62.6	57.3	70.2	-	4.80
NB (2013)	18.0	23.7	9.9	52.9	52.5	53.5	65.4	62.6	69.5	-	3.42
Spectral (2014)	4.5	4.6	4.2	61.5	54.5	71.6	-	-	-	60.7	1.87
SE-WSI-fix	9.8	13.5	4.3	55.1	50.7	61.6	62.9	58.5	69.2	63.0	2.50
SE-WSI-CRP	5.7	7.4	3.2	55.3	49.4	63.8	61.2	56.3	67.9	61.3	2.09
CRP-PPMI	2.9	3.5	2.0	57.7	53.3	64.0	59.2	53.6	67.4	59.2	1.76
WE-Kmeans	4.6	5.0	4.1	51.2	46.5	57.6	58.6	53.3	66.4	58.6	2.54

Table 6: Result on SemEval-2010 WSI task. *80-20 SR* is the supervised recall of 80-20 split supervised evaluation. *FS* is the F-Score of 80-20 split supervised evaluation. *#CI* is the average number of clusters (senses)

baselines according to the above data setting, and compare with reported performance on the same test set from previous papers.

For our sense embedding method, we build two systems: *SE-WSI-fix* which adopts Multi-Sense Skip-gram model (Neelakantan et al., 2014) and assigns 3 senses for each word type, and *SE-WSI-CRP* (Li and Jurafsky, 2015) which dynamically decides the number of senses using a Chinese restaurant process. For *SE-WSI-fix*, we learn sense embeddings for the top 6K frequent words in the training set. For *SE-WSI-CRP*, we first learn word embeddings with word2vec[3], then use them as pre-trained vectors to learn sense embeddings. All training is under default parameter settings, and all word and sense embeddings are fixed at 300 dimensions.

We also design baselines to verify the two advantages of our sense embedding methods. One (*CRP-PPMI*) uses the same CRP algorithm as *SE-WSI-CRP*, but with Positive PMI vectors as pre-trained vectors. The other (*WE-Kmeans*) uses the vectors learned by *SE-WSI-fix*, but separately clusters all the context vectors into 3 groups for each target word with kmeans. We compute a context vector by averaging the vectors of all selected words in the context[4].

4.3.2 Comparing on SemEval-2010

We compare our methods with the following systems: (1) *UoY* (Korkontzelos and Manandhar, 2010) which is the best system in the SemEval-2010 WSI competition; (2) NMF_{lib} (Van de Cruys and Apidianaki, 2011) which adopts non-negative matrix factorization to factor a matrix and then conducts word sense clustering on the test set; (3) *NB* (Choe and Charniak, 2013) which adopts naive Bayes with the generative story that a context is generated by picking a sense and then all context words given the sense; and (4) *Spectral* (Goyal and Hovy, 2014) which applies spectral clustering on a set of distributional context vectors.

[3] `https://code.google.com/p/word2vec/`

[4] A word is selected only if its length is greater than 3, not the target word, or not in a self-constructed stoplist.

Experimental results are shown in Table 6. Let us see the results on supervised recall (80-20 SR) first, as it is the main indicator for the task. Overall, *SE-WSI-fix* performs better than most of the systems, including the best system in the shared task (*UoY*), and *SE-WSI-CRP* works better than *Spectral* and all the baselines. This shows the effectiveness of our methods. Besides, *SE-WSI-CRP* is 1.7 points lower than *SE-WSI-fix*. We think the reason is that *SE-WSI-CRP* induces fewer senses than *SE-WSI-fix* (see the last column of Table 6). Since both systems induce fewer senses than the golden standard which is 3.85, inducing fewer senses harms the performance. Finally, simple as it is, *NB* shows the best performance. However *NB* does not suffer from the domain adaptation problem as it is trained on the training data of the task, and it uses EM algorithm which is generally too slow to benefit from large data. We have other advantages that we train a general model while *NB* learns specific model for each target word.

As for the unsupervised evaluations, *SE-WSI-fix* achieves a good V-Measure score (VM) with a few induced senses. Pedersen (2010) points out that bad models can increase VM by increasing the number of clusters, but doing this will harm performance on both Paired F-score (PF) and SR. Even though *UoY*, NMF_{lib} and *NB* show better VM, they (especially *UoY*) induced more senses than *SE-WSI-fix*. In addition, *SE-WSI-fix* has higher PF than all others, and higher SR than *UoY* and NMF_{lib}. Comparatively *SE-WSI-CRP* has lower VM and induces fewer senses than *SE-WSI-fix*. One possible reason is that the "rich gets richer" nature of CRP makes it conservative for making new senses. But its PF and SR show that it is still a highly competitive system.

To verify the advantages of our method, we first compare *SE-WSI-CRP* with *CRP-PPMI* as their only difference is the vectors for representing contexts. We can see that *SE-WSI-CRP* performs significantly better than *CRP-PPMI* on both SR and VM. *CRP-PPMI* has higher PF mainly because it induces fewer number of senses. The above results prove that using sense embeddings have better performance than using count-based distributional models. Besides, *SE-WSI-fix* is significantly better than *WE-Kmeans* on every metric. As *WE-Kmeans* and *SE-WSI-fix* learn sense centroids in the same vectors space, while the latter performs joint learning. Therefore, the joint learning is better than learning separately.

4.4 Related Work

Kågebäck et al. (2015) proposed two methods to utilize distributed representations for the WSI task. The first method learned centroid vectors by clustering all pre-computed context vectors of each target word. The other method simply adopted *MSSG* (Neelakantan et al., 2014) and changed context vector calculation from the average of all context word vectors to weighted average. Our work has further contributions. First, we clearly point out the two advantages of sense embedding methods: 1) joint learning under the multi-task learning framework, 2) better knowledge representation by discriminative training, and verify them by experiments. In addition, we adopt various sense embedding methods to show that sense embedding methods are generally promising for WSI, not just one method is better than other methods. Finally, we compare our methods with recent state-of-the-art WSI methods on both supervised and unsupervised metrics.

4.5 Conclusion

In this Chapter, we show that sense embedding is a promising approach for WSI by adopting two different sense embedding based systems on the SemEval-2010 WSI task. Both systems show highly competitive performance while they learn a general model for thousands of words (not just the tested polysemous words). we believe that the two advantages of our method are: 1) joint learning under the multi-task learning framework, 2) better knowledge representation by discriminative training, and verify them by experiments.

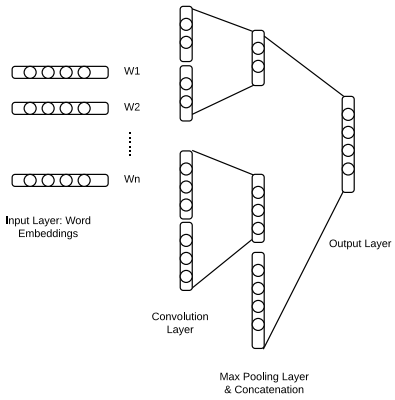

Figure 9: Convolutional Neural Network for context vector modeling.

5 Several Possible Future Research

5.1 Structural Context Representation for Word and Sense Embedding

Previous sense embedding methods always use bag-of-word context representation. However, sometimes we may refer to a structural representation (such as a phrase or a sub-tree) to determine the meaning of the target word. The example below demonstrates a sentence with the polysemous target word *flight*.

*"I took a **flight** with Air Canada"*

By looking at each individual word such as "air" and "Canada" under the bag-of-word scenario, the system may get confused as "air" and "Canada" may refer to some natural scene or pollution issue of Canada. By looking at "Air Canada" together, it is precise that *flight* means "a formation of aircraft in flight".

Here I propose to incorporate phrases, Named Entities and other Syntactic information into context modeling. To incorporate phrases, there are several possible ways: one way is to treat any N-gram words as a phrase, another way is to chunk the training corpus and take a chunk as a phrase. Intuitively, the later way generates more accurate phrases, but suffers the error propagation problem. I propose to adopt the Multiple Sense Skip-Gram (MSSG) model to learn a fixed number of senses.

5.2 Dynamic Word Embedding

One major application for sense embedding is to inference the embedding of a given target word and context. There have been several inference algorithms for sense embedding such as MaxSim, AvgSim and AvgSimC. Previous experiments have shown that there is a huge difference on performance when doing inference with different algorithms. One possible thought is: Can we learn a model that directly and dynamically output the embedding given the target word and context? Here we propose to use Convolutional Neural Network (CNN) and Siamese Network to build a dynamic word embedding model.

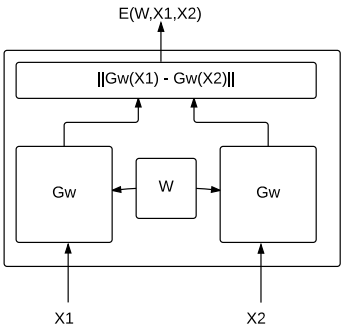

Figure 10: The Siamese network for supervised training.

We use CNN which is shown in Figure 9 to model context representation. Here we take the word embeddings of the context as the input layer, and extract a constant number of N-gram features in the convolution layer for each $N \in [2, 3, 4, 5]$, then use max pooling to randomly drop half of the extracted features, then concatenate the result vector with the target word vector, finally get the output layer which is fully connected with the max pooling layer.

We use Siamese network to train our model. Shown in Figure 10, Siamese network uis a neural network for supervised learning in which the reference (whether X_1 and X_2 are the same) is given. The network takes a pair of instances (X_1 and X_2) each time, then compute the result vectors by putting them into the neural network G_w, then compute the output scalar as the element-wised distance between two output vectors, and finally compute the error by comparing the scalar with the reference. Here G_w is the CNN network in Figure 9, and both G_w share the same parameter W.

For training, we plan to use SemCor corpus as it has reference senses according to WordNet and is large in scale. For each target word, we plan to extract all the tokens of it, and train our model by each pair of tokens.

6 Conclusion

To conclude this paper, In Chapter 2, I introduced several cutting-edge word embedding models which are Skip-gram, Continuous Bag-of-Word and GloVe. I first introduced Skip-gram as it is the most widely used, then introduced the other models by comparing with Skip-gram, finally I reach my conclusion that they actually learn word vectors by maximizing the probability of predicting the context words while minimizing the probability of predicting other words. The prediction probability is defined as the sigmoid function of vector cosine similarity.

In Chapter 3, I introduced several sense embedding models. First of all, I separated these models by the way of learning senses: both clustering-based methods and non-parametric methods define a sense as a cluster of contexts of the target word, while ontology-based methods adopt sense definition from a sense inventory such as WordNet. Clustering-based methods learns a fixed number (K) of senses by clustering all contexts of a word into K groups, thus a hard clustering algorithm (such as K-means) is used. Non-parametric methods use a non-parametric random process to learn a different number of senses ($K(w_i)$) for each word. A Non-parametric process (such as Chinese Restaurant Process) can be seen as a soft clustering algorithm. Ontology-based methods either use a sense inventory as a initialization, then fine-tune the sense vectors on a plain corpus, or directly solve a constrained optimization problem constructed from the sense relations in the sense inventory. In addition, I introduced the state-of-the-art methods in each direction, and finally I showed the evaluation results both on the SCWS dataset (Huang et al., 2012a) and on my own experiment.

In Chapter 4, I introduced a promising way to solve word sense induction (WSI) using sense embedding, and no one has tried that before. I show that sense embedding and WSI are inner related, so sense embedding is naturally applicable to the problem of WSI. In addition, sense embedding has advantages over traditional WSI methods which are: (1) distributed sense embeddings are taken as the knowledge representations which are trained discriminatively, and usually have better performance than traditional count-based distributional models (Baroni et al., 2014), and (2) a general model for the whole vocabulary is jointly trained to induce sense centroids under the multi-task learning framework (Caruana, 1997). I did my experiments on SemEval-2010 WSI task as most previous WSI methods compared their performance on it. My sense embedding-based methods demonstrated highly competitive performance on SemEval-2010.

In Chapter 5, I proposed several possible future works for word and sense embeddings which are: Structural Context Representation for Word and Sense Embedding, and Dynamic Word Embedding.

References

Agirre, E. and Soroa, A. (2007). Semeval-2007 task 02: Evaluating word sense induction and discrimination systems. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, pages 7–12.

Baroni, M., Dinu, G., and Kruszewski, G. (2014). Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland.

Bhingardive, S., Singh, D., V, R., Redkar, H., and Bhattacharyya, P. (2015). Unsupervised most frequent sense detection using word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1238–1243, Denver, Colorado.

Blei, D., Griffiths, T., Jordan, M., and Tenenbaum, J. (2003a). Hierarchical topic models and the nested chinese restaurant process. In *Advances in Neural Information Processing Systems*.

Blei, D., Ng, A., and Jordan, M. (2003b). Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Bordag, S. (2006). Word sense induction: Triplet-based clustering and automatic evaluation. In *EACL*. Citeseer.

Brody, S. and Lapata, M. (2009). Bayesian word sense induction. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 103–111, Athens, Greece.

Caruana, R. (1997). Multitask learning. *Machine learning*, 28(1):41–75.

Chen, P., Ding, W., Bowes, C., and Brown, D. (2009). A fully unsupervised word sense disambiguation method using dependency knowledge. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 28–36, Boulder, Colorado.

Chen, X., Liu, Z., and Sun, M. (2014). A unified model for word sense representation and disambiguation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1025–1035, Doha, Qatar.

Choe, D. K. and Charniak, E. (2013). Naive Bayes word sense induction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1433–1437, Seattle, Washington, USA.

Goldwater, S. and Griffiths, T. L. (2007). A fully Bayesian approach to unsupervised part-of-speech tagging. In *Proceedings of ACL*, Prague.

Goyal, K. and Hovy, E. H. (2014). Unsupervised word sense induction using distributional statistics. In *COLING*, pages 1302–1310.

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Huang, E., Socher, R., Manning, C., and Ng, A. (2012a). Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 873–882, Jeju Island, Korea.

Huang, E. H., Socher, R., Manning, C. D., and Ng, A. Y. (2012b). Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics.

Jauhar, S. K., Dyer, C., and Hovy, E. (2015). Ontologically grounded multi-sense representation learning for semantic vector space models. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 683–693, Denver, Colorado.

Korkontzelos, I. and Manandhar, S. (2010). Uoy: Graphs of unambiguous vertices for word sense induction and disambiguation. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 355–358, Uppsala, Sweden.

Kågebäck, M., Johansson, F., Johansson, R., and Dubhashi, D. (2015). Neural context embeddings for automatic discovery of word senses. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 25–32, Denver, Colorado.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Lau, J. H., Cook, P., McCarthy, D., Newman, D., and Baldwin, T. (2012). Word sense induction for novel sense detection. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 591–601, Avignon, France.

Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Li, J. and Jurafsky, D. (2015). Do multi-sense embeddings improve natural language understanding? In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1722–1732, Lisbon, Portugal.

Manandhar, S., Klapaftis, I., Dligach, D., and Pradhan, S. (2010). Semeval-2010 task 14: Word sense induction & disambiguation. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 63–68, Uppsala, Sweden.

Meilă, M. (2003). Comparing clusterings by the variation of information. In *Learning theory and kernel machines*, pages 173–187. Springer.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Navigli, R. and Crisafulli, G. (2010). Inducing word senses to improve web search result clustering. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 116–126, Cambridge, MA.

Neelakantan, A., Shankar, J., Passos, A., and McCallum, A. (2014). Efficient non-parametric estimation of multiple embeddings per word in vector space. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1059–1069, Doha, Qatar.

Pedersen, T. (2010). Duluth-wsi: Senseclusters applied to the sense induction task of semeval-2. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 363–366, Uppsala, Sweden.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar.

Purandare, A. and Pedersen, T. (2004). Word sense discrimination by clustering contexts in vector and similarity spaces. In Ng, H. T. and Riloff, E., editors, *HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 41–48, Boston, Massachusetts, USA.

Reisinger, J. and Mooney, R. J. (2010). Multi-prototype vector-space models of word meaning. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 109–117, Los Angeles, California.

Rong, X. (2014). word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Rothe, S. and Schütze, H. (2015). Autoextend: Extending word embeddings to embeddings for synsets and lexemes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1793–1803, Beijing, China.

Shaoul, C. and Westbury, C. (2010). The westbury lab wikipedia corpus.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2006). Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581.

Tian, F., Dai, H., Bian, J., Gao, B., Zhang, R., Chen, E., and Liu, T.-Y. (2014). A probabilistic model for learning multi-prototype word embeddings. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 151–160, Dublin, Ireland.

Van de Cruys, T. and Apidianaki, M. (2011). Latent semantic word sense induction and disambiguation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1476–1485, Portland, Oregon, USA.

Wang, J., Bansal, M., Gimpel, K., Ziebart, B., and Yu, C. (2015). A sense-topic model for word sense induction with unsupervised data enrichment. *Transactions of the Association for Computational Linguistics*, 3:59–71.

Xiong, D. and Zhang, M. (2014). A sense-based translation model for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1459–1469, Baltimore, Maryland.

Yao, X. and Van Durme, B. (2011). Nonparametric bayesian word sense induction. In *Proceedings of TextGraphs-6: Graph-based Methods for Natural Language Processing*, pages 10–14.